

NPSCS-91-015

NAVAL POSTGRADUATE SCHOOL
Monterey, California

com
(2)

AD-A243 268



DTIC
ELECT
DEC 11 1991
S D



**AN OBJECT-ORIENTED APPROACH
TO SECURITY POLICIES AND THEIR ACCESS CONTROLS
FOR DATABASE MANAGEMENT**

David K. Hsiao

September 1991

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

91-17544



NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared for and funded by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:



DAVID K. HSIAO
Professor of Computer Science

Reviewed by:

Released by:



ROBERT B. MCGHEE
Chairman
Department of Computer Science



P.J. MARTO
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-91-015			5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School		6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Research Laboratory	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			7b. ADDRESS (City, State, and ZIP Code) Code 5542, Naval Research Laboratory Washington, D.C. 20375	
8a. NAME OF FUNDING SPONSORING ORGANIZATION Naval Postgraduate School		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER OM&N Direct Funding	
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	TASK NO.
			PROJECT NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) An Object-Oriented Approach to Security Policies and Access Controls for Database Management				
12. PERSONAL AUTHOR(S)				
13a. TYPE OF REPORT		13b. TIME COVERED FROM 7/91 TO 9/91		14. DATE OF REPORT (Year, Month, Day) 1991 September
15. PAGE COUNT 33				
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	Object-Oriented Data Model, Inheritance, Covering, The Need-to-Know Policy, The Multilevel Security Policy.	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
<p>The constructs of the object-oriented data model seem to be good candidates for the specification of the need-to-know and multilevel security policies and their respective access control requirements. This report demonstrates such specifications. The implication of this demonstration may be profound, since for the first time multiple security policies and their respective access controls may be realized and supported in a single object-oriented database management system.</p>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS HPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL David K. Hsiao			22b. TELEPHONE (Include Area Code) (408) 646-2253	22c. OFFICE SYMBOL CSHq

Preface

There are two articles in this report: Part One and Part Two. Although these two parts are related, each is on a different security policy and a different object-oriented construct. By treating them separately, it is hoped that there is an in-depth discussion of the subject matters involved. However, the introduction of Part One may be served as the introduction of the report. The concluding remarks of Part Two may be considered as the conclusion of the report. Thus, the report should be read in sequel, i.e., Part One first and Part Two next.

In 1990 while preparing a tutorial on object-oriented approach to database management, the author was impressed by the rich and powerful constructs provided in the object-oriented data model (oodm). These constructs were intended for specifying database organizations, processing requirements, and integrity constraints. However, they soon dawned on the author that they may be also good for specifying security policies and access control requirements. The present report is an exercise of such specifications. The argument for specifying the security policies and access control requirements are many. They are articulated in the report. The argument against specifying security policies and access control requirements are few. There are nevertheless indicated in the report.

The Information Technology Division (ITD) of the Naval Research Laboratory (NRL) is examining the oodm and its role in integrity constraints. Our examination of the oodm and its role in security issues complements their examination nicely. The work reported herein is supported by funds provided by ITD of NRL. The author would like to thank Dr. Carl Landwehr of NRL for the support. Thanks also due to Dr. Landwehr's colleagues at ITD for some stimulating discussions and technical exchanges on the topical matters. Last but not the least, I would like to thank our German Visiting Scholar, Daniel A. Keim, for proof-reading the report and for commenting on the material.

Accession For	
NTIS	CRAGI
DTIC	TAB
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Availability Special
A-1	

**The Relationship of Data Models and Security Requirements:
Part One -
The Object-Oriented Data Model and the Need-to-Know Policy***

David K. Hsiao

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
U.S.A.

Abstract

This is the first of two papers on the issues of data models and their capabilities or incapacities to support various security policies and their access control requirements. It is important to design a data model which accommodates a security policy and its access control requirements. Otherwise, we must extend the data model and modify its database system. If a data model supports a specific security policy and its access control requirements, then we must make certain that there is also a database system which supports the database characterized by the data model. It is always easier to come up a new data model; it is much harder to build a database system for the data model.

As monopoly data models and their database systems proliferate, issues of resource consolidation and data sharing appear. In a large organization which requires many different security policies and their access control requirements, there will be a large number of heterogeneous database systems each of which supports a different security policy and its unique access control requirements for one of the heterogeneous databases. Such proliferations raise the questions: How can we consolidate various databases, system software and hardware, and support personnel so that we may conserve the support and maintenance cost? How can we interface various database systems so that data sharing among heterogeneous databases may be facilitated without compromising the autonomy of individual security policies and their access control requirements in the organization?

An apparent solution to aforementioned issues is to have (1) a data model which allows different security policies and access control requirements to be specified for their corresponding databases and to have (2) a database system which supports databases characterized by the data model. In this paper, we show that the object-oriented data model (oodm) allows the need-to-know policy and its access control requirements to be specified for an object-oriented database. In the second paper, we will show that the oodm allows the multilevel security policy and its access control requirements to be specified for another object-oriented database. The interesting consequence of these specifications is that both policies over their respective databases may be supported by an object-oriented database system (OODBMS) which meets our goal for resource consolidation and data sharing. Thus, an OODBMS is also a multipolicy DBMS which requires no extension of the oodm and no modification of the OODBMS for supporting different policies.

*The work reported here is supported in part by funds provided from NRL.

1. INTRODUCTION

Conventional data models are *not* intended for supporting security policies and their access control requirements over their respective databases so modeled. They are designed for querying and manipulation of modeled databases. Here, we provide some backgrounds on the past effort to modify or adopt certain data models for the support of specific security policies and their access control requirements. We also point out the limitations and pitfalls of these modifications and adoptions.

1.1. The Restriction to a Specific Security Policy and an Ad Hoc Access Control Mechanism

A case in point can be found in the use of the relational data model to support the *need-to-know* policy and the use of the *view* mechanism in specifying the access control requirements for the policy. The *need-to-know* policy has never been clearly stated in the relational model. Even the *view*, a construct necessary for any access control to relations, is not found among the constructs of the relational data model. Some refer to it as a *derived relation* [1]. However, the notion of derived relations has not been incorporated into the relational data model either. Instead, it is relegated to the data language of a particular database system which supports relational databases. For instance, IBM SQL/DS allows the *view* to be a language construct of SQL.

On the other hand, INGRES, a relational database system, also supports the *need-to-know* policy. However, its data language, QUEL, does not use the *view* mechanism for access control. Instead, it uses the *query-modification* mechanism [2]. Thus, the notion of a *view* does not exist in QUEL. By default, all conventional database systems support the *need-to-know* security policy and various ad hoc means to facilitate its access control requirements. The conventional database user is therefore restricted to the same security policy and an ad hoc access control mechanism of the database system employed.

1.2. The Need of Other Security Policies and Access Control Requirements

A large organization may have different security policies for different databases. For example, as a large organization, the U.S. DoD requires the use of the *multilevel security* policy for accessing classified information and the *need-to-know* policy for accessing unclassified information. It is not surprising that in DoD data management a large number of IBM SQL/DS and INGRES are being employed for the purpose of supporting unclassified databases of payrolls and inventories, for example. On the other hand, if we place classified information as databases in a database system, then the database system must support the *multilevel security* policy and its access control requirements. There are two approaches to meet this need. They are expounded in the following sections.

A. To Extend a Conventional Data Model for the Purpose of Accepting a New Security Policy

The notion of *polyinstantiations* of the relational tuple [3] is an effort to extend the relational data model for supporting the *multilevel security* policy and its read-down and write-up requirements. Although the introduction of *polyinstantiations* to the relational model is a rigorous mathematical

exercise and the extension, i.e., the extended data model, is *not* a relational model in that a polyinstantiated relational database can not be placed in a conventional relational database system such as IBM SQL/DS or INGRES for access and manipulation operations. In fact, to display its capability for polyinstantiations of tuples, the extended data model uses an Entity-Relationship database system for such displays. We all know that the E-R data model [4] is not the relational data model.

Consequently, a conventional relational database system must also be modified and extended for the polyinstantiation, i.e., for polyinstantiated relational databases. The modification and extension are applied not just to the database system itself, but also to the data language whether the language is SQL or QUEL. Retrofitting an existing system and its ad hoc access control mechanism to accommodate a new security policy and its access control requirements is a messy task. It may result in a system less reliable and secure than building a new system from scratch. Further, we have design and implementation methodologies for building new systems, but we do not have methodologies for retrofitting old systems.

B. To Find a New Data Model for the Support of the New Security Policy

Ideally, the intrinsic property of a data model supports a given security policy. The database designer merely has to recognize that the data model can characterize the database which meets the access control requirements of the security policy. There is no need for the database designer to modify the data model. Nor is there a need for the database designer to retrofit the database system for the new database.

Such an attempt has taken place in the *attribute-based data model*. It has been found that the equivalence relation of the attribute-based data model can be used to characterize the compartmentalization of classified data under the multilevel security policy [5]. Further, the access control mechanism of the attribute-based database system [6] supports the access control requirements of the multilevel security policy such as the read-down operation [7].

However, the attribute-based database system is experimental. There is no commercial attribute-based database system. To have an attribute-based database system with production quality, we may have to retrofit the experimental attribute-based database system or build a new attribute-based database system from scratch. Again, we fall into the same dilemma as we have encountered in Section A.

1.3. The Limitation of Monomodel and Monopolicy Database Systems

From above discussions, we learn that it is possible to build different database systems for different data models, each of which supports a distinct security policy and its access control requirements. Since each of these database systems has its own data model and supports its unique security policy, we term them *monomodel and monopolicy database systems*. In a large organization where a number of security policies and access control requirements over different databases is required, there will be a number of monomodel and monopolicy database systems. The proliferation of large

number of monomodel and monopoly database systems and their databases in an organization creates the following issues: resource consolidation and data sharing. We address these two issues in the following sections separately.

A. The Issue of Resource Consolidation

Monomodel and monopoly database systems and their databases proliferate in an organization in order to support multiple security policies and their different access control requirements, correspondingly. There is an ever-increasing addition of new database software, hardware and personnel to accommodate and maintain the new policies, databases and systems. Consequently, there is the need of *resource consolidation*, i.e., to consolidate all the database software into one database system, all the database hardware into one computer system, and all the support personnel into one team. Resource consolidation enables the organization to exercise centralized controls and to minimize redundant resources which are ideal for a security-conscious and cost-conscious organization.

However, conventional monomodel and monopoly database systems are immuned from resource consolidation. The *multimodel and multilingual database system* [8, 9], on the other hand, may lend itself for the support of multiple policies and their access control requirements in the same system which is therefore an ideal candidate for resource consolidation. The arrival of a multimodel and multilingual database system will be at least a decade away [10, 11]; meanwhile, we need a ready solution to the issue of resource consolidation.

B. The Issue of Data Sharing

Ideally, monomodel and monopoly database systems scattered at different localities in an organization can be interconnected via a net so that a database of a local database system may be accessed by a remote user and controlled by the local security policy and its access control mechanism of the database system. Such is really the issue of data sharing among *distributed heterogeneous databases and systems*. Presently, we have no solution towards data sharing among distributed heterogeneous databases and systems. Thus, present-day heterogeneous database systems and their databases are run in isolation with no data sharing among themselves. We need a ready solution to the issue of data sharing among monomodel and monopoly database systems now.

2. AN EXPERIMENTATION WITH THE OBJECT-ORIENTED DATA MODEL (OODM)

The object-oriented approach to programming has long preceded the object-oriented approach to database management. Whereas an object-oriented programming language may be viewed by a programmer as just another programming language, the oodm to a database practitioner is not like any of the conventional data models. A conventional data model tends to provide a few simple and limited *structures* for the database designer to organize data for the database. For example, there are the table of the relational data model, the hierarchy of the hierarchical data model, and the net-

work of the Codasyl data model. The oodm, instead of providing certain data structures (e.g., tables, hierarchy, and network) for the organization of data into structured entities of a database, provides *properties* to be defined and maintained for data. In fact, the lack of static data structures in the oodm enables many detractors of the oodm to object the notion of an object as being too structureless and vague.

However, as far as our concern for supporting different security policies and their access control requirements, the oodm is superb. Further, we begin to have object-oriented database systems of production quality. If we can capture all the security policies and their access control requirements into object-oriented databases, then we can support these object-oriented databases in an object-oriented database system (OODBMS). The issues of resource consolidation and data sharing will disappear, since we will be operating in a *homogeneous database* environment (i.e., all databases are object-oriented), instead of a heterogeneous database environment based on many different data models. In the following sections we elaborate on our observation.

2.1. Two Properties of the OODM and their Relationship to Different Security Policies

The concept of *inheritance* is intrinsic to the relationship of one object class with another. Let B be an object class. B is said to be an object subclass of an object class A, if B *inherits* all the property from A. The inheritance is meant for B to have all the *attributes* and *actions* of A. We are going to illustrate in this paper that the inheritance property of the oodm supports the need-to-know policy and its access control requirements. In our illustration, we then introduce the terminology and definition of the inheritance property of the oodm.

On the other hand, the concept of *covering* is intrinsic to the relationship of one object class with subsets of another object class. Let A and B be two object classes. We say that A and B have a *covering* relationship; more specifically, we say that A is a *cover* of B if every object of A corresponds to a subset of objects of B. The *correspondence* is a mapping f which determines for an object, a , of A all the objects, b 's, of the subset $f(a)$ of B such that $f(a) = b$ for every one of those b 's. We will illustrate in our second paper that the covering property of the oodm supports the multilevel security policy and its access control requirements. In the second paper, we will also introduce the terminology and definition of the covering property of the oodm.

2.2. The Use of an Object-Oriented Database System to Support Databases under Different Security Policies.

From the previous section, we learn that the oodm has properties which support at least two different, well-known security policies and their access control requirements. In other words, object-oriented databases may be specified, generated and protected under different security policies and their access control requirements. All these databases can be managed by an OODBMS which of course upholds the policies and requirements for their respective databases.

This OODBMS is of course a *monomodel* and *multipolicy* database system. As a homogeneous database system, the issues of resource consolidation and data sharing disappear.

3. THE OODM'S INHERITANCE IN SUPPORT OF THE NEED-TO-KNOW POLICY

We first introduce the notion of inheritance as it is defined over the objects of an object-oriented database. We then show how the inheritance can be used to define the need-to-know policy and its access control requirements for a given database.

3.1. Objects, Object Classes and Object Hierarchies

The most fundamental notion and construct in an object-oriented approach to database management are the notion and construct of an object. An *object* is a data aggregate of a class; i.e., it has some value of the class. To coincide the class of values of a database object with the type of values of a programming object, some database practitioners also use the term, *type*, in lieu of the term, class, and the term, *instance*, instead of the term, aggregate. In this paper, we use one set of terminology only.

Objects may be simple or complex. Examples of *simple* objects can be found in the *primitive* objects where each class of primitive objects has an unique and basic data class such as integers, floating-point numbers, or character strings. More specifically, in the object class, INTEGERS, every primitive object, i.e., an integer, has a primitive value, i.e., the integer proper. Formally, we write down the object class as follows:

Object class:	INTEGERS
Attribute:	Integer

The new term, attribute, denotes the property, i.e., the value class, of the object. In this example, the literal, INTEGERS, is the name of the object class and the literal, Integer, names the placeholder for all its attribute values. We may have the following primitive object class:

Object class:	FLOATING-POINTS
Attribute:	Floating-point number

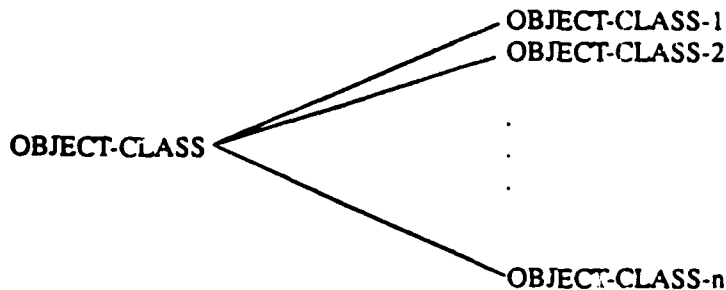
An object is *complex* if it is formed with other primitive and/or existing objects as its attributes. Therefore, a complex object class consists of complex objects of the same attributes. The following is a specification of a complex object class made of two primitive object classes:

Object class:	NUMBERS
Attribute 1:	INTEGERS
Attribute 2:	FLOATING-POINTS

The use of capital letters in naming the attributes indicating that these are names of object classes, since we have consistently used the capitals for such purposes in earlier specifications. We also provide some spaces to separate the specification of an object class from the specification of its attributes. In general, a complex object class, OBJECT-CLASS, has the following specification:

Object class:	OBJECT-CLASS
Attribute 1:	OBJECT-CLASS-1
Attribute 2:	OBJECT-CLASS-2
	.
	.
Attribute n:	OBJECT-CLASS-n

Graphically, they are depicted as follows:



The above structure of a complex object class depicts a two-level hierarchy of object classes. In general, multiple *levels* are possible in a *class hierarchy*.

3.2. Actions

An *action* is an operation or a function that is defined for an object class and can be performed on objects of the object class. In fact, the only way to operate on an object of an object class is by means of the operation or function defined for the object class. Here, we have the major departure from the traditional database management and conventional programming. In traditional database management, transactions operate on data aggregates of a database. These transactions are written by the application programmer and introduced at different times and for different applications. They are managed separately from the database by the DBMS. Further, they are *not* transactions written in terms of specific operations or functions pre-defined for individual data aggregates.

In conventional programming, operations or functions in a program are defined solely for a set of built-in data structures. They are not meant for data structures in other programs. In programming, we bring data structures into individual programs. On the other hand, in the OODBMS, we bring operations or functions into a database and associate them with individual object classes of the database. This is in the opposite way of programming. It is also different from traditional database management where few operations or functions are pre-defined for, restricted to, and incorporated into specific data aggregates of a database.

What are the benefits of incorporating an operation or a function, i.e., an action, into objects of an object class? It defines a *legitimate* operation or function to be performed on objects of a given ob-

ject class. It also provides a *standard* interface for the application programmer to form a complex object of interfaced objects, and write a transaction with standard operations or functions of the interface on interfaced objects. By restricting accesses to and manipulations of objects to the legitimate operation (or function) and standard interface, the *operational and data integrity and operational and data security* of objects can be assured. We will illustrate these notions and constructs with examples in later sections.

Here, we introduce the notion and construct of simple and complex actions. *Primitive* actions are simple, built-in actions. They are provided by the OODBMS. Examples of the primitive actions are as follows:

Read -	the input-data-for-an-object function,
Get -	the get-the-object-identifier-of-an-object function,
Find -	the get-the-object-by-its-identifier function,
Include -	the include-the-object-of-an-object-class-into-a-set-of-objects-of-the-object-class function,
Exclude -	the exclude-the-object-of-an-object-class-from-a-set-of-objects-of-the-object-class function,
Check -	the check-the-membership-of-the-object-of-an-object-class-in-a-set-of-objects-of-the-object-class function,
Update -	the modify-the-object-of-an-object-class function,
Post -	the post-error function;
End -	the end-of-an-action function.

Complex actions are made of primitive and/or existing actions on the basis of some action composition rules. These rules are supported by the OODBMS. Examples of *action composition rules* are as follows:

The *sequencing* rule - sequencing a number of functions or operations by placing semicolons (;) between two adjacent functions or operations.

The *conditional* rule - the IF-THEN-ELSE construct with functions or operations as conditions of the construct.

Let us form, for example, several complex actions by using the primitive actions given previously and the action composition rules proposed above.

Add-user:	Read user data; Get user identifier; Check validity; IF data ok THEN Include user ELSE Post error; End
Drop-user:	Get user identifier; Find user; IF user exists THEN Exclude user ELSE Post error; End
Change-password:	Get user identifier; Read new password; Find user; IF user exists THEN Update password ELSE Post error; End

In summary, we have specified *three* complex actions in terms of *nine* simple actions and *two* action composition rules. Actions are also referred to as *methods*. In this paper we use the term, action, over the other.

3.3. Another Look at Objects and Object Classes

Actions are always specified for an object class, whether they are simple or complex. Therefore, they may also be considered as a part of the *property* of the object class. In specifying an object class, we may now name actions of the object class as its additional property. Considering user records as objects of an object class and complex functions defined for the change of user passwords as actions of the object class, we may have the following specification:

Object class:	USERS
Attributes:	USER IDENTIFIER USER NAME USER PASSWORD
Actions:	Add-user Drop-user Change-password

Attributes of the example are complex, since they are capitalized and defined in terms of primitive

or existing object classes which are defined elsewhere. The three complex actions have been defined in terms of the nine primitive ones and two rules given earlier.

In general, we have the following specification of an object class:

Object class:	(the name of an object class)
(Property list:	list attributes first, then actions next.)
Attributes:	OBJECT-CLASS-1 OBJECT-CLASS-2 . . . OBJECT-CLASS-n
Actions:	Action-1 Action-2 . . . Action-m

It is important to note that the *real definition* of an object class includes not only its attributes, but also its actions. This is the definition we will be using in the subsequent discussion of and reference to the notion and construct of object classes in object-oriented database management.

3.4. The Object-Class Hierarchy and its Inheritance

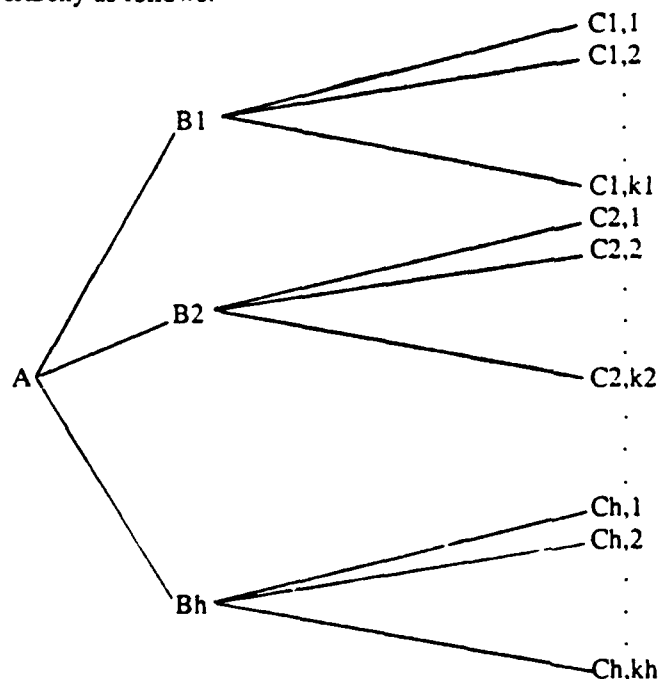
So far we have focused on the intrinsic property of the object class individually, i.e., the *common* attributes and actions of all the objects of a given class. However, we have not considered the relationship of one object class with another object class. The concept of *inheritance* is intrinsic to the relationship of one object class with another. Let B be an object class. B is said to be an *object subclass* of an object class A, if B inherits all the property from A. The inheritance is meant for B to have all the attributes and actions of A. The converse is not true, if B may have additional attributes and/or actions which are not in the property of A.

If all the attributes and all the actions of the object class A are *common* to all its object subclasses, B1, B2, . . . , Bh and there may be some property of Bi (for $i = 1, \dots, h$) which is *not* common to A, then, in this case, we have a two-level *object-class hierarchy* of A and Bi.

The definition of inheritance also suggests two kinds of inheritance: the *data inheritance* where B inherits all the attributes of A, and the *operational inheritance* where B inherits all the actions of

A. The data inheritance allows the application of strong classification over objects in the hierarchy. The operational inheritance enables a common set of legitimate operations or functions to be applied to objects in the hierarchy. Largely due to these two inheritances *data and operational integrities and data and operational securities* of an object class A and its subclasses B1 through Bh are upheld in the OODBMS. In other words, object-class hierarchies facilitate data and operational integrities as well as data and operational securities among its object classes in their respective hierarchies.

Let C_{ij} be object subclasses of B_i where j ranges from 1 through k_i (i.e., k_1 for C_{1j} , k_2 for C_{2j} , . . . , k_h for C_{hj}). Since B_i are subclasses of the object class A, we have a three-level object-class hierarchy as follows:



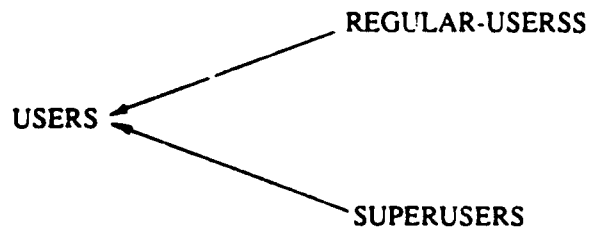
In general, an object-class hierarchy may have multiple *levels*. Consider the two object classes below. We note that some attributes and actions are new here whose primitives are assumed to be defined elsewhere. We observe that both object classes, REGULAR-USERS and SUPERUSERS, have three common attributes and three actions of the object class, USERS, depicted earlier. Thus, both REGULAR-USERS and SUPERUSERS inherit from USERS.

Conversely, the object class, USERS, does not inherit from either REGULAR-USERS or SUPERUSERS, since (1) neither REGULAR-USER LIST nor SUPERUSER LIST is in the attribute list of USERS and (2) neither the verify-regular-user's-password action nor the monitor-superuser's-access action is in the action list of USERS. Actually, either condition (1) or condition (2) will be sufficient to invalidate the converse; there is no need to have both conditions.

Object class:	REGULAR-USERS
Attributes:	USER IDENTIFIER USER NAME USER PASSWORD REGULAR-USER LIST
Actions:	Add-user Drop-user Change-password Verify-regular-user's-password

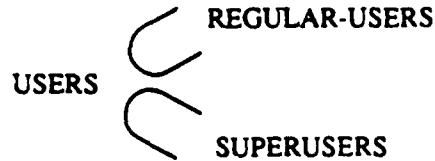
Object class:	SUPERUSERS
Attributes:	USER IDENTIFIER USER NAME USER PASSWORD SUPERUSER LIST
Actions:	Add-user Drop-user Change-password Monitor-superuser's-access

Graphically, they are depicted in a two-level object-class hierarchy below:



Due to space limitation, we have not listed for each object class its property in terms of attributes and actions.

Either the arrow notation (i.e., the *is-pointed-by* relation) or the subset notation (i.e., the *is-a-subset-of* relation) has been used in the graphical representation of an object-class hierarchy, since either notation is an *is* relation. Thus, the above graph can also be found as follows:



3.5. Generalization and Specialization vs. Common-Subset and Unique-Superset

The use of the subset notation in the hierarchy is most revealing. It indicates that every property of **USERS** is a property of **REGULAR-USERS** and **SUPERUSERS**. Thus, **USERS**' property become the common property, i.e., common subset, of **REGULAR-USERS** and **SUPERUSERS**. If all the property of an object class becomes the *common* subset of properties of many object classes, then we say that the object class is a *generalization* of these many object classes. Conversely, each of these many object classes is a *specialization* of the generalization, since it has some property *uncommon* to some others and *unique* to itself. Inheritances in an object-class hierarchy always induce generalizations and specializations. In the sample hierarchy, we say that **USERS** are a generalization of **REGULAR-USERS** and **SUPERUSERS**. Conversely, either **REGULAR-USERS** or **SUPERUSERS** are a specialization of **USERS**.

In the set-theoretic terminology, the generalization is the *common-subset* property. The specialization is the *unique-superset* property. Thus, in referring to the same example, we say that the property of **USERS** is a subset of the property of either **REGULAR-USERS** or **SUPERUSERS**. Conversely, either the property of **REGULAR-USERS** or the property of **SUPERUSERS** is a superset of the property of **USERS**. In practice, professionals in programming languages tend to use terms, generalization and specialization, whereas professionals in database management tend to use their equivalent terms, common-subset property and unique-superset property, since database operations are mostly set-oriented operations.

3.5. The Need-to-Know Policy and its Access Control Requirements

We now apply the object-oriented notions and constructs introduced in the last five sections to the secure database application in which the need-to-know policy is upheld and its access control requirements are facilitated. Every database has an *owner*. The owner of a database is asked (1) to specify the database as an object class, called the *owner object class*, and (2) to specify for each user of the database an *user object class* as a subset of the owner object class. In other words, the owner object class is the common superset of all the subsets as characterized by their corresponding user object classes. Thus, the owner object class is a specialization of each user object class which is in turn a generalization of the owner object class.

In general, we may have the following specification where the owner object class has m attributes and n actions as its property. Database attributes are usual *value-types* such as **NAME**, **EMPLOYEE#**, **AGE**, **POSITION**, **SALARY**, and **DEPARTMENT**. We denote them with attribute-1, attribute-2, . . . , attribute- m . Database actions are *primary database operations* such as retrieve, insert, update, delete, and merge/join. We denote them with action-1, action-2, . . . , action- n . Thus,

the owner object class can be depicted as following:

Object class: OWNER
Attribute-1: NAME
Attribute-2: EMPLOYEE#
.
.
.

Attribute-m: AGE
Action-1: Retrieve
Action-2: Insert
.
.
.

Action-n: Merge/Join

On the other hand, for each user object class the i attributes and j actions specified for the user are respective subsets of the m attributes and n actions specified for the owner. More specifically, for the user k we have the following specification:

Object class: USER-K
Attribute-k1: (one of the m attributes)
Attribute-k2: (other one of the m attributes)
.
.
.

Attribute-ki: (another one of the m attributes)
Action-k1: (one of the n actions)
Action-k2: (other one of the n actions)
.
.
.

Action-kj: (another one of the n actions)

The important observation in the above specification is that (1) there are as many as k users who have accesses to the same database owned by the owner; (2) each user's access to the database is controlled individually by an owner's specification for the user; (3) in each specification the k_i attributes accessible to user k form a subset of the owner's m attributes; (4) in the same specification the k_j actions exercisable by user k forms a subset of the owner's n actions; (5) since k_i 's and k_j 's may be different among themselves, all k users may be subject to different access controls for different needs to know. Graphically, we have the following:

Object-class: USER-1

Attribute-1,1: (one of the m attributes)
Attribute-1,2: (other one of the m attributes)

Attribute-1,i1: (another one of the m attributes)
Action-1,1: (one of the n actions)
Action-1,2: (other one of the n actions)

Action-1,j1: (another one of the n actions)

Object-class: USER-2

Attribute-2,1: (one of the m attributes)
Attribute-2,2: (other one of the m attributes)

Attribute-2,i2: (another one of the m attributes)
Action-2,1: (one of the n actions)
Action-2,2: (other one of the n actions)

Action-2,j2: (another one of the n actions)

Object-class: USER-K

Attribute-k,1: (one of the m attributes)
Attribute-k,2: (other one of the m attributes)

Attribute-k,ik: (another one of the m attributes)
Action-k,1: (one of the n actions)
Action-k,2: (other one of the n actions)

Action-k,jk: (another one of the n actions)

Object-class: OWNER

Attribute-1: NAME
Attribute-2: EMPLOYEE#

Attribute-m: AGE
Action-1: Retrieve
Action-2: Insert

Action-n: Merge/Join

3.7. The Use of the Object-Class Hierarchy for Access Controls

Let us first discuss the use of attributes of an object-class hierarchy for access controls. We then elaborate on the use of the actions in access controls.

A. Attributes of an Object-Class Hierarchy

The attributes specified for a user in the user's object class is equivalent to the attributes specified for the user in the user's view of a relational database. However, unlike the oodm where a user object class is specified as a subset, i.e., a generalization, of the owner's object class in an object-class hierarchy, a user view is specified in an ad hoc manner by way of a relational data language, say, SQL. Further, the hierarchical relationship of a relation and its views between the owner and all the users of the owner's database is not spelled out. On the other hand, in the oodm the object-class hierarchy is clearly spelled out as we have depicted one in the previous section. Thus, the object-class hierarchy is a natural and effective means to specify access control requirements for the need-to-know policy.

B. Actions as Access Privileges and Queries in an Object-Class Hierarchy

The actions specified for a user in the user's object class subsumes the *access privileges and queries* specified in the user's view of a relational database. Access privileges include read, update, append, delete, etc. An access query is a logical expression of predicates defining the part of a database of which the user is granted some access privileges. These specifications are again done in an ad hoc manner in a relational data language. Further, it is never clear to the owner or the user how the controlled accesses are carried out by the relational database system (RDBMS). Consider the following two examples:

With a given access query will the RDBMS accept the user's online query, modify the user's online query with the access query, retrieve from the database all the data satisfying the modified query, and route all the retrieved data to the user? Here, the RDBMS performs a *query modification* which enables those and only those authorized data to be retrieved and brought to the real memory.

Or, with a given access query will the RDBMS accept the user's online query, retrieve from the database all the data satisfying the online query, sort out from the real memory all the retrieved data satisfying also the access query, and route the sorted-out data to the user? In this case, the RDBMS performs a *filtering process* which filters out all the unauthorized data for the user.

Obviously, the user and owner prefer the query modification over the filtering process in access controls, since the former is more efficient than the latter. Nevertheless, few users or owners can discover the access control mechanism of a given RDBMS. On the other hand, into the OODBMS the owner can introduce the exact access control mechanism for the support of the need-to-know policy. Since a user can only write a database transaction against a database in terms of the actions specified in the user object class for the database, one of the pre-specified actions can be the only access mechanism. This action is obviously introduced by the owner of the database, since the

owner is the one who has provided the specification of the object-class hierarchy in the first place. In this way, the owner can determine for the user the exact access control mechanism. The subset property, i.e., the generalization property, requires that all the users (whose object classes are in the hierarchy) to use the same and common access control mechanism, i.e., the common action defined for them. It also precludes a user from introducing a different access control mechanism, i.e., a new action, since the new one would not form a subset of the set of owner's actions.

C. Actions as Integrity Constraints and Security Requirements in an Object-Class Hierarchy

Minimally, actions in a user object class are authorized access privileges, access queries, and primary database operations for the user. However, the role of actions in an object-class hierarchy is much larger and more embracing. Two extended controls - one for data integrity and the other for data security - can also be facilitated with the use of actions.

Operational and data integrity is our concerns over the correctness and validity of attribute values. To this end, for vital values we specify integrity constraints. In a conventional DBMS an *integrity constraint* is usually a transaction defined over a data aggregate. The transaction will be triggered, i.e., executed, by the DBMS whenever the data aggregate is accessed. The transaction obviously checks the access operation and makes certain that the correctness and validity of data being accessed are upheld. However, the use of triggers and integrity constraints requires the user to learn, in addition to the data model of the database, the data language and the system functions of the DBMS. Further, it is difficult to write an integrity constraint in a data language, since data languages, unlike programming languages, have limited capabilities in checking values.

Actions, on the other hand, may be included in the user object class in a hierarchy by the owner as integrity constraints. Since these are the only legitimate actions that the user can exercise, there is no need of some triggering mechanism to activate them. The user is compelled to use these actions for the user's transaction. Further, an action is a program in the true sense of programs of a programming language. The only requirement is that there are some *couplings* between the constructs of a programming language and the constructs of the oodm via the compiler of the programming language.

Operational and data security is our concerns over the granularity and access of attribute values. Obviously, the owner would like to have controls over the amount of data a user may access and the kind of operations a user may perform on the amount. *Granuals*, i.e., amount of data in a data aggregate, may be as large as the entire database or as small as an attribute value. Actions as *aggregate functions* can be used to delineate the granuals. The conventional DBMS provides aggregate functions. However, the owner of a conventional DBMS has no way to compel a user of the owner's database to access data via certain aggregate functions. In owner object-class hierarchy, the user has no other way, but the way specified for the user object class in the user's actions. These actions may be aggregate functions dictated by the owner.

Actions can also be used to provide 'high-level' access operations. Whereas read, write, append and others are considered primitive access operations, retrieve, delete, update, insert and others are considered as primary database operations. Although all these operations must be secured, they are nevertheless considered low-level. Low-level access and database operations are typically pro-

vided and secured in a conventional DBMS. What we would like to secure are high-level access and database operations which are introduced and dictated by the owner of a database. For example, an action may be introduced by the owner to further authenticate the user of the owner's database. Whenever the user accesses the database, the action demands additional passwords or identifications for the entry to the database. Thus, this action serves as a security 'gate-keeper' of the owner's database. For another example, an action may be introduced by the owner to audit certain pending sets of output of a user of the owner's database. If these sets of output may compromise the security of the database due to certain accesses, the audit trails may be used for postmortem analyses. Here, the action serves as a security 'trail-follower' of user's accesses.

Whether for exotic integrity constraints or for high-level security requirements, actions are the ideal means for the owner to introduce them into user object classes in the owner object-class hierarchy.

4. CONCLUDING REMARKS

In this part one, we have argued that the conventional data models and their database systems are not adequate for operational and data securities of the database. We have also argued that the extension of an existing data model, no matter how popular the model is, for upholding a specific security policy and its access control requirements requires the modification or retrofitting of an existing database system - which is also an undesirable and insecure effort. Lastly, we have argued in the introduction that the support of separate policies and their different access control requirements on various database systems creates an environment of heterogeneous database systems and databases. This environment prevents controlled sharing of heterogeneous databases and effective consolidation of all the heterogeneous system software, hardware, and personnel.

We have argued for the experimentation with the oodm. The owner object-class hierarchy and the inheritance property of the oodm support the need-to-know policy and its access control requirements naturally and effectively. To this end, we have introduced the necessary definitions and enough examples to illustrate the use of the object-class hierarchy and the inheritance for such access controls. In the next experimentation, i.e., in the part two of this paper, we will introduce the *covering property* of the oodm. The covering property supports the multilevel security policy and its access control requirements. It is hoped that these experimentations may show that the oodm is a multipolicy data model. Since an object-oriented database system can support the oodm, the same database system can uphold multiple security policies and their corresponding access control requirements. Thus, this is a homogeneous database system with homogeneous databases, despite their different policies and requirements. By nature, homogeneous databases and their database system provide data sharing and resource consolidation. The issues with heterogeneous databases and systems disappear in the OODBMS.

5. REFERENCES

- [1] Codd, E. F., *The Relational Model for Database Management: Version 2*. Addison-Wesley, 1990.

- [2] Stonebraker, M. R., and Wong, E., "Access Control in Relational Database Management Systems by Query Modification," *Proceedings of ACM National Conference*, 1974.
- [3] Denning, D. E., Lunt, T. F., Schell, R. R., Heckman, M., and Shockley, W. R., "A Multilevel Relational Data Model," *Proceedings of IEEE Symposium on Security and Privacy*, 1987.
- [4] Chen, P. P.-S., "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transaction on Database Systems*, 1,1, 1976.
- [5] Hoppenstand, G. S., and Hsiao, D. K., "Secure Access Control with High Access Precision: An Efficient Approach to Multilevel Security," *Database Security, II: Status and Prospects*, (Editor, C. E. Landwehr) North-Holland, 1989.
- [6] Demurjian, S. A. Hsiao, D. K., and Menon, J., "A Multi-backend Database System for Performance Gains, Capacity Growth and Hardware Update," *Proceedings of the Second IEEE International Conference on Data Engineering*, 1986.
- [7] Hsiao, D. K., Kohler, M. J., and Stround, S. W., "Query Modifications as a Means of Controlling Accesses to Multilevel Secure Databases," *Database Security, IV: Status and Prospects*, (Editors, S. Sajodia and C. Landwehr) North-Holland, 1991.
- [8] Demurjian, S. A., and Hsiao, D. K., "The Multi-Model Database System," *Proceedings of International Phoenix Conference on Computers and Communications*, March 1989.
- [9] Demurjian, S. A., and Hsiao, D. K., "Towards a Better Understanding of Data Models through the Multilingual Database System," *IEEE Transactions on Software Engineering*, SE-14, 7, 1988.
- [10] Hsiao, D. K., and Kamel, M. N., "Heterogeneous Databases: Proliferations, Issues and Solutions," *IEEE Transactions on Knowledge and Data Engineering*, KDE-1,1,1989.
- [11] Hsiao, D. K., "Databases and Database Systems in the 21-st Century," *SIAM Proceedings of Symposium on Very Large Scale Computation*, 1991.
- [12] Hsiao, D. K., "The Object-Oriented Database Management - A Tutorial on its Fundamentals," (unpublished) 1990.

**The Relationship of Data Models and Security Requirements:
Part Two -
The Object-Oriented Data Model and the Multilevel Security Policy***

David K. Hsiao

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
U.S.A.

Abstract

This is the second of two papers on the issues of data models and their capabilities or incapacities to support various security policies and their access control requirements. We have argued in the first paper [1] that the conventional data models are monopolicy data models each of which can only support a single security policy and its access control requirement. Consequently, there is a proliferation in a large organization of monopolicy database systems which enforce multiple security policies and their corresponding access control requirements, respectively. Such a proliferation in the organization of heterogeneous databases and systems creates the problem of data sharing among various heterogeneous databases and of resource consolidation of all the database system software, hardware, and personnel. Further, a popular database system may not support a specific security policy and its access control requirement. Retrofitting the popular database system for the support of a specific security policy and its access control requirement may be time-and-effort-consuming. It may also be error-prone. On the other hand, there are experimental database systems each of which supports a number of security policies and their access control requirements. Such multipolicy database systems are ideal for data sharing and resource consolidation. Unfortunately, few such experimental multipolicy database systems are of production quality. They cannot be used in an organization for production work. The question is therefore whether or not we can find a production-quality database system whose data model can support multiple security policies and their corresponding access control requirements without retrofitting the data model and its database system?

The object-oriented database system (OODBMS) seems to be such a good candidate. The object-oriented data model (oodm) has built-in constructs which, although primarily intended for various integrity constraints, can be used to specify various security policies and their access control requirements. In the Part One (again see [1]), we have shown that the oodm supports the need-to-know policy and its access control requirement over the unclassified data. In this paper, i.e., Part Two, we would like to show that the oodm supports also the multilevel security policy and its access control requirement over the classified data. Together, Parts One and Two indicate that a typical OODBMS is a multipolicy database system. Further, the OODBMS is coming out the experimental stage into the production world.

*The work reported herein is supported in part by funds provided from NRL.

1. INTRODUCTION

In the introduction of Part One [1], we have argued against the use of conventional or experimental data models and their underlying database systems for the support of different security policies and their access control requirements. We have also argued for the need of a data model and its production-quality database system which can support a number of security policies and their access control requirements. We are not going to repeat those arguments here.

What we have also demonstrated in Part One is the use of *inheritance* of the object-oriented data model (oodm) to specify the need-to-know policy and its access control requirement over the unclassified data. In this part, i.e., Part Two, we would like to show that the other construct of the oodm, known as the *covering*, may be used to specify the multilevel security policy and its access control requirement over the classified data. Both the inheritance and the covering constructs of the oodm are sufficiently powerful and general; we believe that other security policies and their access control requirements may also be specified in them.

With our demonstrations in Part One and in Part two, we hope to show that the oodm is indeed a multipolicy data model whose object-oriented database system (OODBMS) may support both classified and unclassified data, exercise respective access controls, and support different security policies. Thus, in a large organization the proliferation of monopolicy data models and their database systems will no longer be necessary. The issues of data sharing and resource consolidation of heterogeneous, monopolicy databases and systems will not be there. What we have in the organization is a homogeneous database system, i.e., OODBMS, whose data model, i.e., oodm, allows various security policies and access control requirements to be specified for various databases and to be supported by the same database system.

2. THE OODM'S COVERING

Although inheritance is considered to be basic in the oodm, covering may be considered by some as optional in the oodm. However, as far as our concern the presence of covering in the oodm is essential. Whereas inheritance deals with the need-to-know policy and its access control requirement, covering is particularly useful and natural for us to specify the multilevel security policy and its access control requirement.

The reason that inheritance is not that all-inclusive is due perhaps to its inability to handle the mapping of an object of an object class to a set of objects of another object class. Inheritance typically maps an object class to one or more other object classes. Thus, covering may be thought as object-to-class mappings, whereas inheritance as class-to-class mappings. We can be more specific about them in the following sections.

2.1. Objects, Object Classes and Object Hierarchies

Definitions of and motivations for the object, object class, and object hierarchy have been given in Part One [1], we are not going to repeat them here. For those who have not yet read Part One, the following short definitions are given: Objects are instances of an object class; each object class has

a list of attributes and another list of actions; attributes may be primitive (e.g., made of values of the same type) or complex (made of primitives); actions may also be primitive (e.g., built-in programs) or complex (i.e., made of primitive and/or existing actions on the basis of some action composition rules). If all the attributes and all the actions of the object class A are common to all the object classes, B1, B2, . . . , Bh (called subclasses of A) and there may be some attribute or action of Bi (for $i = 1, \dots, h$) which is not common to A, then we have a two-level object-class hierarchy of A and Bi. In general, an object-class hierarchy may have multiple levels.

2.2. The Object-Class Hierarchy and its Inheritance

In the aforementioned object-class hierarchy, we say that Bi *inherits* all the property from A. The inheritance is meant for Bi to have all the attributes and actions of A. From the viewpoint of mappings, there is a mapping of A into each Bi. It is also a mapping of an object class into another object class, i.e., *object-class-into-object-class* mapping. Further, it is an *one-to-many* mapping, since there is only one A and many B's. On the other hand, covering induces a different kind of mapping as follows.

2.3. The Covering Construct and its Mapping

Let C and D be two object classes. We say that C is a *cover* of D if every object of C corresponds to a subset of objects of D. These subsets of D need not partition D; they are certain subsets of all the subsets generated for the objects of D. Mathematically, all the subsets of D form the *power set* of D, i.e., $P(D)$. The *correspondence* is a mapping f which determines for an object, c , of C all the objects, d 's, of the subset $f(c)$ of D such that $f(c) = d$ for every one of those d 's.

In the mathematical notation, we have

$$f: C \rightarrow P(D).$$

In the set notation, we have the subset $f(c)$ of D as

$$\{ d \mid f(c) = d \text{ where } c \text{ is in } C \text{ and } d\text{'s are in } D \}.$$

C is termed the *cover object class* and, correspondingly, D the *member object class*.

Graphically, we depict the cover class C and its member class D as

$$C \rightarrow P(D).$$

It is important to note that the covering corresponds an object, not an object class, to a subset of the power set of another object class. Any subset of an object class can be considered as an inheritance of the object class, since attributes and actions of the subset are included in the attributes and actions of the object class. Thus, the mapping is from-object-into-class, i.e., *object-into-object-class* mapping. Covering is also known as aggregation just as actions are also known as methods.

Every object of C may form a *singleton*, i.e., an object class consisting of the object singly. The mapping may now be considered as from-object-class-into-object-class, i.e., *object-class-into-object-class* mapping as well. From the viewpoint of object classes, the mapping corresponds many singleton classes with a single object class D, the mapping is therefore *many-to-one*.

3. THE MULTILEVEL SECURITY POLICY AND ITS ACCESS CONTROL REQUIREMENT

The *multilevel security policy* applies only to classified data. If we place classified data as a database in a database system, then the database should be compartmentalized on the basis of the multilevel security classification. The *multilevel security classification* requires that every piece of classified data is to be identified by its *classification level* such as top secret, secret, or confidential. These classification levels establish a strict security hierarchy with, for example, top secret 'above' the secret level, and confidential 'below' the secret level. *Compartmentalization* enables classified data of the same level to be stored, accessed and controlled separately from classified data of other levels.

The multilevel security policy requires that users of classified data to be grouped into security clearances. *Clearance levels* of users are identical to classification levels of data.

Further, the multilevel security policy requires the database system to support specific and unique access operations on its compartmentalized database. They are the read-down and write-up operations. The *read-down operation* allows a user of the classified database to read all the data whose classifications are either below or identical to the clearance of the user. The *write-up operation* allows a user of classified data to write into the database a piece of classified data whose classification is either above or identical to the clearance of the user.

It is important to note that the multilevel security policy and its access control requirement are established without due regard to a specific database system. Thus, when we decide to place classified data as database in a database system, we must make certain that the database system can uphold the multilevel security policy and enforce its access control requirement. It is the main thesis of this paper that (1) the covering of the oodm can characterize the compartmentalization of classified data efficiently and effectively, thereby supports the multilevel secure database; (2) the OODBMS can uphold the policy and enforce read-down and write-up operations effectively and efficiently, thereby supports the multilevel secure users.

4. THE OBJECT-ORIENTED REALIZATION OF THE MULTILEVEL SECURE DATABASE

One of our major objections to the use of the relational data model for the multilevel secure database is the inability of relational database systems to provide any compartmentalization of classified data. The elegant and complex mathematical extension of the relational data model via polyinstantiation [2] allows classified data to be compartmentalized on the basis of their classifications. However, the polyinstantiated database is *not* a relational database. Consequently, *no* existing relational database system can support the relational-like polyinstantiated database. Obviously, we look into other data models.

4.1. The Attribute-Based Data Model and its Equivalence Relation

In [3, 4] we have shown that the use of the attribute-based data model (abdm) to compartmentalize classified data on the basis of their classifications is straightforward. It requires no mathematical extension of the abdm. Thus, the multilevel secure database for classified data is indeed an attribute-based database which can be supported by any attribute-based database system. However, the only attribute-based database system in existence is experimental which can not be used widely as a production system. Nevertheless, we may ask ourselves the question: what is the property of the abdm that supports the compartmentalization of classified data effectively and efficiently?

There is the property of the abdm known as the *clustering* property of the abdm. Clustering is an equivalence relation (i.e., the mathematical relation that induces equivalence classes) which partitions the database into mutually exclusive sets of records, known as *clusters*. There is the direct correlation between the notion of clusters and the notion of compartments. In other words, if we can construct an equivalence relation in the oodm, then we can compartmentalize classified data as we have clustered the attribute-based data. In the following sections we show how an equivalence relation can be introduced into the covering construct of the oodm.

4.2. Equivalence Relations and their Coverings of the Object-Oriented Classified Data

As in the definition of covering in Section 2.3, any covering relationship involves two object classes, C and D , and a mapping f from objects of C to members of $P(D)$, the power set of D . Thus, for each object of C the function f corresponds to a subset of objects of D . However, these subsets of D need not partition D . For our application, we must partition D . In fact we must also partition C . We therefore in the following first specify the object class D and its partitioning property. We then specify the object class C and its partitioning property. Finally, we specify the function f . As it turns out, we need another function g .

A. The Object Class of Classified Data

The object class, CLASSIFIED-DATA, consists of objects, i.e., *classified documents*, one for each distinct classification and content. Since the multilevel security classification of classified data is itself an equivalence relation (we leave it to the reader to verify this equivalence relation), it partitions all the subsets $P(\text{CLASSIFIED-DATA})$ of the object class CLASSIFIED-DATA into mutually exclusive categories of objects (i.e., equivalence classes of classified documents) one for each classification. For example, all the top-secret data are in the category of top-secret objects; all the secret data are in the category of secret objects; all the confidential data are in the category of confidential objects; and so on. In each category, there are all the subsets of objects of the same classification. For example, in the top-secret category, there are one-piece sets of top-secret documents, two-piece sets of top-secret documents, three-piece sets of top-secret documents, and so on.

It is important to note that the mathematical notion of equivalence classes is different from the object-oriented notion of object classes. Whereas a subset as a member of an equivalence class cannot be a member of another equivalence class, subsets of an object class in a covering (e.g., the

object class D in Section 2.3) need not partition the object class. Further, an equivalence class consists of some distinct members (e.g., in our case, subsets of $P(D)$) which are different from any other equivalence class in the same equivalence relation. Finally, an object class can only consist of objects whereas an equivalence class may consist of anything as long as they are partitioned by the equivalence relation.

Intuitively, the classification of the classified data partitions not only individual classified documents with different classifications into mutually exclusive sets of documents, but also groups all the documents of various sizes and of the same classification into the same set. Thus, the multilevel security classification is the equivalence relation which partitions all the subsets of $P(\text{CLASSIFIED-DATA})$. The number of partitions of classified data is equal to the number of classification levels. For the first time, we can associate the term partition with term compartment. Objects of the **CLASSIFIED-DATA** or subsets of $P(\text{CLASSIFIED-DATA})$ are compartmentalized, so that all the objects or object subsets in a compartment have the same classification.

B. The Object Class of Cleared Users

The object class, **CLEARED-USERS**, consists of objects, i.e., *user profiles*, one for each distinct clearance and user id. As it turns out the number of clearance levels is equal to the number of classification levels. Thus, the clearance can be used as an equivalence relation to partition the user profiles into mutually exclusive sets of objects such that all the users in a set have the same clearance. Since user ids in objects of **CLEARED-USERS** are distinct and there is an one-to-one correspondence between user profiles and user ids, the user id can be used as an equivalence relation to partition further each set of user profiles into individual profiles with their own unique ids.

With the user id, we can uniquely identify an object, i.e., user profile, of the object class, **CLEARED-USERS**. With only the user's clearance, we can uniquely identify a set of objects whose objects all have the same clearance.

C. Mappings of an Object of Cleared Users to a Subset of the Power Set of Classified Data

Mathematically, we specify the function f in the following :

$$f: \text{CLEARED-USERS} \rightarrow P(\text{CLASSIFIED-DATA})$$

where **CLEARED-USERS** is the cover object class and **CLASSIFIED-DATA** is the member object class. To each object, i.e., user profile, of **CLEARED-USERS**, we correspond subsets of objects, classified documents, of **CLASSIFIED-DATA** such that each subset of classified documents has a common classification level below or identical to the user's clearance level in the user profile.

Consider a sample multilevel secure database with four classification levels: top-secret, secret, confidential, and unclassified. The entire database is called the object class, **CLASSIFIED-DATA**. The power set, $P(\text{CLASSIFIED-DATA})$, is partitioned by the classification into four and only four sets of subsets:

{subsets of objects of classified documents with the top-secret classification},

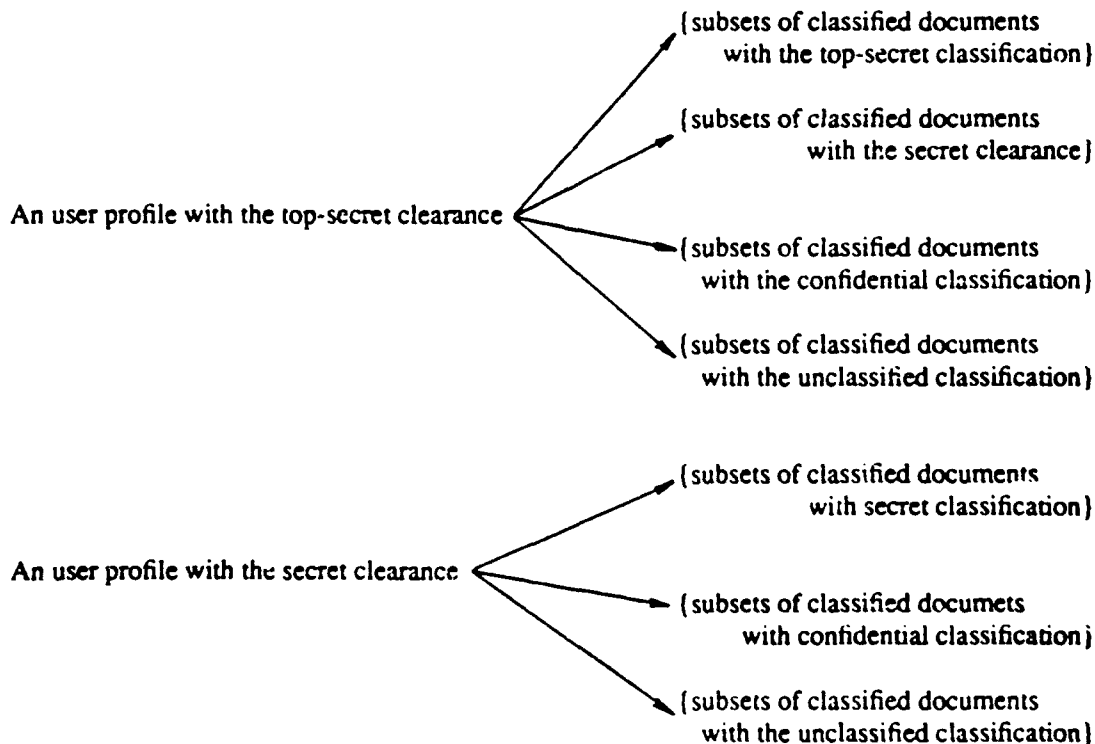
{subsets of objects of classified documents with the secret classification},

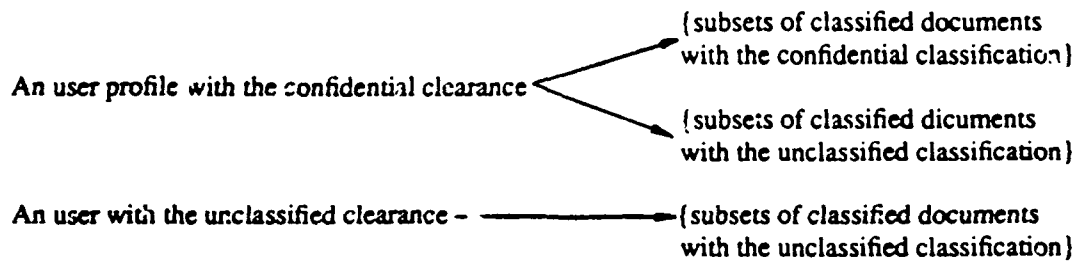
{subsets of objects of classified documents with the confidential classification},

{subsets of objects of classified documents with the unclassified classification}.

The important observations are that (1) the above four subsets are mutually exclusive and (2) each is actually a set of subsets, not just a subset.

Now, consider an object of the object class, CLEARED-USERS. The object is obviously a user profile which contains an attribute, termed CLEARANCE-LEVEL. If its attribute value is top-secret, then f maps the object to all the aforementioned four partitioned sets of $P(\text{CLASSIFIED-DATA})$. If its attribute value is secret, then f maps the object to all the remaining three partitioned sets of $P(\text{CLASSIFIED-DATA})$, except the first one with top-secret classification. If its attribute value is confidential, then f maps the object to the two partitioned sets of $P(\text{CLASSIFIED-DATA})$ with confidential and unclassified classifications, respectively. Finally, if its attribute value is unclassified, then f maps the object to the only partitioned set with the unclassified classification. Graphically, we have the following mapping, f , from an object of object-class, CLEARED-USERS, to four subsets of $P(\text{CLASSIFIED-DATA})$:

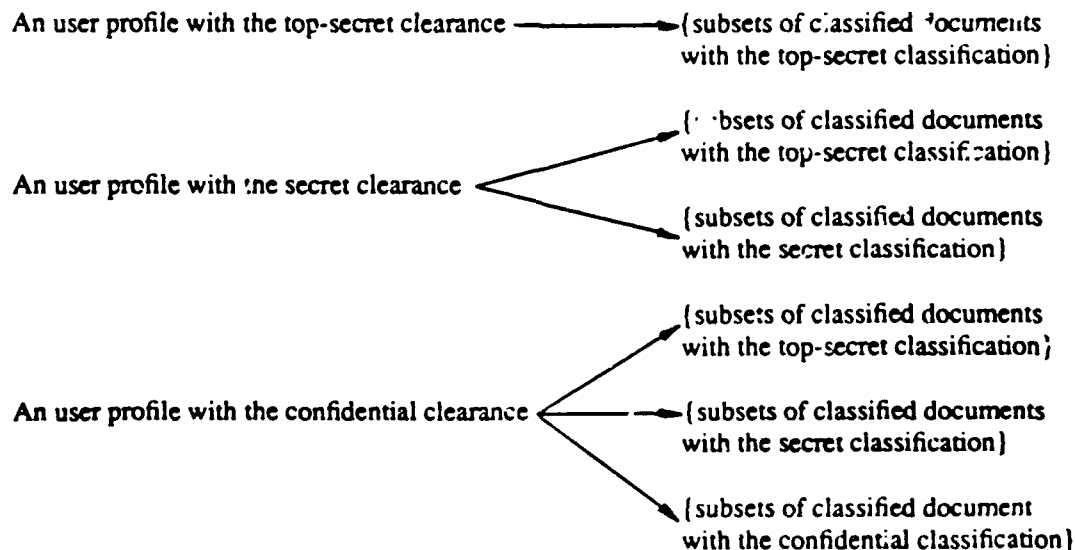


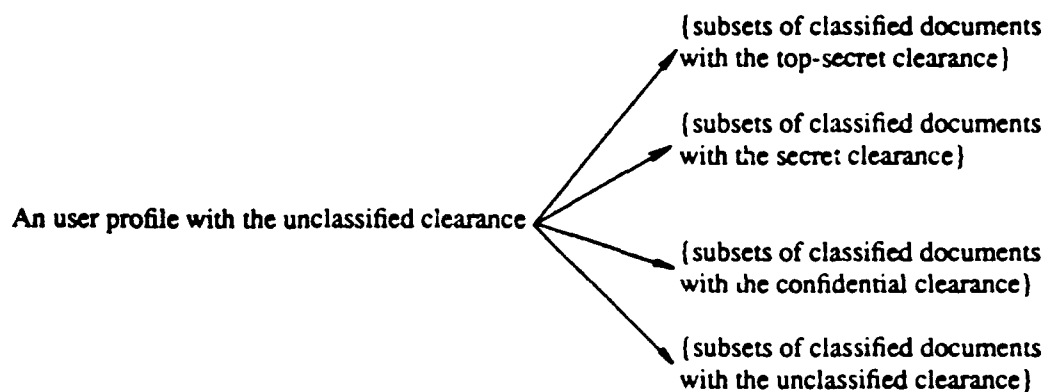


From the above specification, it is clear that the mapping f facilitates one of the two multilevel secure operations, the *read-down* operation, since all the classified data which are cleared for the user for read accesses have been mapped to the user profile. We need another mapping to facilitate the write-up operation. Consider g in the following:

$g: \text{CLEARED-USERS} \rightarrow P(\text{CLASSIFIED-DATA})$

where **CLEARED-USERS** is again the cover object class and **CLASSIFIED-DATA** is again the member object class. To each object, i.e., user profile, of **CLEARED-USERS**, we correspond subsets of objects, classified documents, of **CLASSIFIED-DATA** such that each subset of classified documents has a common classification level above or identical to the user's clearance level in the user profile. Using the same example for f , we now illustrate the specification of g . Consider a user profile in the object class **CLEARED-USERS**. If its attribute value is top-secret, then g maps the object to the only partitioned set of $P(\text{CLASSIFIED-DATA})$ with the top-secret classification. If its attribute value is secret, then g maps the object to the two partitioned sets of $P(\text{CLASSIFIED-DATA})$ with top-secret and secret classifications, respectively. If its attribute value is confidential, then g maps the object to the three remaining partitioned sets of $P(\text{CLASSIFIED-DATA})$, except the partitioned set with the unclassified classification. Finally, if its attribute value is unclassified, then g maps the object to all the four partitioned sets of $P(\text{CLASSIFIED-DATA})$.





From the above specification, it is also clear that the mapping g facilitates the other multilevel secure operation, i.e., the *write-up* operation, since all the classified data which are cleared for the user for write accesses have been mapped to the user profile. It is important to note that in a write access the write-up operation means an insertion. It does not mean to write over the existing classified data.

D. The Read-Down and Write-Up Operations as Actions of the Classified-Data Object Class

In the previous section we have specified two functions, f and g , for two covering hierarchies, respectively. Let us call the covering hierarchy defined by f the *f-covering hierarchy* of the cleared user to classified data and the covering hierarchy defined by g the *g-covering hierarchy* of the cleared user to classified data. Obviously, the f -covering hierarchy is established for the access control of the read-down operation of the multilevel security policy over classified data. For this reason, it is also called the *read-down hierarchy*. On the other hand, the g -covering hierarchy is established for the access control of the write-up operation of the multilevel security policy over the same classified data. We also term it the *write-up hierarchy*.

The actually actions which perform either the read-down operation or the write-up operation are embedded in the object class, CLASSIFIED-DATA. Let us call the read-down action the *f-action* and write-up action the *g-action*. Obviously, the f -action can only be carried out in the f -covering hierarchy and the g -action can only be carried out in the g -hierarchy. For these reasons, we refer to the two actions as the access control operations and the two hierarchies as the access control hierarchies for the multilevel security policy.

5. CONCLUDING REMARKS ON INHERITANCE AND COVERING OF THE OODM

We all know that the presence of a data model is for the user to characterize and generate a database on the basis of constructs of the data model so that the database may be supported on the database system of the data model. As far as we are concerned, there are two major constructs in the oodm: inheritance and covering. Let us review their capabilities in the following sections and then con

sider some new issues.

5.1. On their Capabilities

There are five outstanding capabilities of the oodm and its OODBMS. We expound them in the following paragraphs. In our exposition of these capabilities, we attempt to relate them to the security issues of our concern or to other data models and their database systems in existence.

A. Naturalism

We have demonstrated that the inheritance construct can be used to characterize and generate an objected-oriented database for the need-to-know policy and its access control requirement. The demonstration was published in the Part One [1]. In that demonstration the use of inheritance for the compliance of the policy and for the enforcement of its access control was straightforward. Thus, inheritance is a *natural* construct in the oodm to characterize and generate objected-oriented databases for database applications under the need-to-know policy and its access control.

On the other hand, the covering construct of the oodm is used herein to characterize and generate an object-oriented database for classified data under the multilevel security policy and its access control requirement. This demonstration is presented in the Part Two, i.e., this paper. The use of covering for the compliance of the policy and for the enforcement of its access control is also straightforward. Thus, covering is a *natural* construct in the oodm to characterize and generate object-oriented databases for classified data and for data accesses under the multilevel security policy and its access control.

B. Generality

Of a conventional data model, the constructs are intended *specifically* for a class of applications. For examples, the relational data model and its constructs are intended for tables and table managements. Thus, the relational database system is good for table-oriented applications [5]. The network data model and its constructs are intended for keeping track of inventories. Thus, the network database system is used for inventory-control applications [6]. The hierarchical data model and its constructs are intended for managing production or design assemblies. Thus, the hierarchical database system is good for keeping track of product or design assemblies [7]. Of course, we can always use one data model and its constructs to 'emulate' the constructs of another data model so that applications intended for the other model may be supported on the data model and database system on hand. Emulations create complications and violate the rule of naturalism as expounded in the previous section.

Neither inheritance nor covering has been constructed specifically for either the need-to-know policy or the multilevel-security policy, respectively. Nevertheless, they provide constructs which naturally characterize the respective databases. This may be due to the *generality* of the object-oriented constructs, since they are not intended for specific applications. The vested generality in

the oodm may allow us to characterize and generate additional databases for new policies and their access control requirements.

C. Flexibility

A number of equivalence relations is needed in a secure database system. These equivalence relations are used to partition either the users or the data for the security purpose of forming distinct user groups or separate data aggregates. In the attribute-based data model [3], a built-in equivalence relation based on the Cartesian product of descriptors is made available to the database designer for the purpose of partitioning the attribute-based database into clusters. Unfortunately, there is no attribute-based database system in production; one can only support an attribute-based secure database experimentally [4]. On the other hand, none of the conventional data models provides any equivalence relation. The possibility for a conventional database system to support well-compartmentalized databases and well-segregated user groups is not there.

The oodm does not have built-in equivalence relations either. However, the covering construct lends itself easily for a definition of an equivalence relation, since the Cartesian product can easily be applied to all the subsets of the power set of the database object class. For an example, in our demonstration earlier, the power set has been $P(\text{CLASSIFIED-DATA})$. Thus, the oodm is sufficiently *flexible* for the database designer to introduce the desired equivalence relation to partition $P(\text{CLASSIFIED-DATA})$.

D. Multipolicy

Conventional database systems are all monopoly database systems. By *monopoly* we mean that a database system can only support a single security policy and its access control requirement. Most of the conventional database systems support the need-to-know policy and its access control requirement. To support another security policy and its access control requirement, the conventional user must acquire another monopoly database system. The proliferation of various monopoly database systems in a large organization hinders data sharing and resource consolidation, since the interoperability issues among heterogeneous databases supported on their respective heterogeneous database systems have not been resolved [8].

On the other hand, the OODBMS is a *multipolicy* database system; i.e., in a single database system several security policies and their access control requirements are supported. As a homogeneous database system with a single data model, the issues of interoperability among heterogeneous database systems do not exist. There is also no proliferation of new database systems, since we continue to utilize the OODBMS on hand. In Part One and Part Two of this paper, we have demonstrated that the OODBMS is at least *bipolicy*.

E. Practicality

Unlike the experimental attribute-based database system [4], there are object-oriented database

systems of production quality, i.e., *in practice*. We can specify several databases - one for each policy and its access control requirement - and have these databases supported on an OODBMS for various data security applications under different security policies and their access control requirements.

5.2. On other Issues

Despite aforementioned merits, a number of new issues must be resolved before the viability of the oodm and its OODBMS can be sustained. These issues are expounded in the following paragraphs.

A. The Lack of a Standard OODM

Unlike a conventional data model which was proposed and specified by a single source, object-oriented approaches to database management came from many sources. Thus, there are many object-oriented data models, instead of a single, therefore, standard oodm.

There is a historical and technical reason for the proliferation of object-oriented data models. The notion of object-oriented approaches to programming has been initiated in the programming language community before the notion of object-oriented approaches to database management become vogue in the database community. Since programming involves data or data sets, there are object-oriented constructs to deal with them. However, these data or data sets are primarily memory-bound and temporary, not permanent disk-based databases. Database practitioners attempt to extend and expand the object-oriented programming constructs to object-oriented database constructs. Consequently, there are many and different extensions and expansions which preclude a standard oodm.

It is not clear whether or not every oodm is endowed with both inheritance and covering. Further, of those with both inheritance and covering it is also not clear whether or not the constructs provided for either of them are identical to the ones we use in our realization of the need-to-know policy or the multilevel security policy, since there is no single and authoritative specification of all the constructs of inheritance and covering.

B. The Lack of a Standard OODBMS

A conventional database system of a conventional data model is either (1) an industrial standard based on the common specification of its data language and system architecture put out by an industrial and/or user consortium or (2) a de facto standard put out by an industrial giant. For instances, all commercial network database systems use the CODASYL data manipulation language, i.e., CODASYL-DML, and have same system features. All the commercial relational database systems use SQL and have same system features. In object-oriented database systems, there is the lack of not only a standard oodm but also either an industrial or de facto standard for the OODBMS. Consequently, there are many different object-oriented database systems.

It is therefore not clear that among those object-oriented database systems having both inheritance and covering whether or not they can provide enough language constructs and system features for us to specify (1) access control requirements as they are dictated by their respective policies, (2) processing requirements as dictated by respective applications, and (3) system features as dictated by necessary interfaces with the existing operating and computer system.

5.3. On their Prospects

Perhaps, the only way to resolve the aforementioned issues is to try a number of object-oriented data models and their database systems for our security applications. Since the use of object-oriented approaches to the secure database management is a new kind of applications, it may even open up additional applications for these models and database systems. We should not bypass the object-oriented data models and their database systems for our secure database applications, due to their variety. We should sort them out on the basis of our needs and expectations. Thus, the prospect for the oodm and OODBMS to support secure database management should be promising.

6. REFERENCES

- [1] Hsiao, David K., "The Relationship of Data Models and Security Requirements: Part One - The Objected-Oriented Data Model and the Need-to-Know Policy," included in this report.
- [2] Denning, D. E., Lun, T. F., Schell, R. R., Heckman, M., and Shockley, W. R., "A Multilevel Relational Data Model," *Proceedings of IEEE Symposium on Security and Privacy*, 1987.
- [3] Hoppenstand, G. S., and Hsiao, David K., "Secure Access Control with High Access Precision: An Efficient Approach to Multilevel Security," *Database Security, II: Status and Prospects*, (Editor, C. E. Landwehr), North-Holland, 1989.
- [4] Hsiao, David K., Kohler, M. J., and Stroud, S. W., "Query Modifications as a Means of Controlling Accesses to Multilevel Secure Databases," *Database Security, IV: Status and Prospects*, (Editors, S. Jajodia and C. E. Landwehr), North-Holland, 1991.
- [5] Codd, E. F., *The Relational Model for Database Management: Version 2*, Addison-Wesley, 1990.
- [6] CODASYL, "Report of the Data Description Language Committee," *Information Systems*, 3, pp. 247-320, 1978.
- [7] Geller, J. R., *IMS Administration, Programming, and Data Base Design*, John Wiley & Sons, 1989.
- [8] Hsiao, David K., and Kamel, M. N., "The Multimodel and Multilingual Approach to Interoperability of Multidatabase Systems," *Proceedings of First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 7-9, 1991.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5100	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943-5100	1
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	1
David K. Hsiao Code CSHq Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	10
Chief of Naval Research 800 N. Quincy Street Arlington, VA 22302-0268	1
Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Mr. Lynwood Sutton (Code 8322) Naval Ocean Systems Center San Diego, CA 92152	1

Mr. Al Wells & Ms. Leah Wong
Code 443
Naval Ocean Systems Center
San Diego, CA 92152

2

Ms. Doris Mlezco
Code 9033
Naval Pacific Missile Test Center
Point Mugu, CA 93042

1

Messrs. Hank Steubing & Chuck Koch
Code 50C
Naval Air Development Center
Warminster, PA 18974

2

Capt. O'Neal, USN
Code 630
Naval Security Group Command
Washington, DC 20390

1

Mr. John Cambel
National Computer Security Center
1500 Savage Drive
Fort Meade, MD 20755

1

Mr. John Hooder
ITAC
Building 166
Washington Navy Yard
Washington, DC 2003

1

Dr. Carl Landwehr
Code 5542
Naval Research Laboratory
Washington, DC 20375

2